DB Migration

Michel Casabianca
casa@sweetohm.net

Sur Github: https://github.com/c4s4/db_migration

Cet outil permet de gérer les migrations d'une base de données. On doit écrire les scripts de migration et les disposer dans des répertoires par version et par plate-forme. Par exemple, pour migrer la PRP vers la version 1.2.3, on placera le script dans le fichier `1.2.3/prp.sql`.

Le script de migration maintient deux tables : - `_scripts` liste les scripts passés - `_install` liste les installations réalisées.

Par conséquent, le script sait quels scripts doivent être passés pour migrer vers la version cible et l'on n'a pas à connaître la version de départ de la migration.

Conventions de nommage

Les données d'initialisation de la base de données se trouvent dans un répertoire appelé `init`. Dans ce répertoire doit se trouver un script appelé `all.sql` qui sera passé sur toutes les plateformes et qui contiendra les schémas de la base de données. On pourra y placer des scripts `itg.sql`, `prp.sql` et `prod.sql` par exemple contenant les données par environnement.

Les scripts d'initialisation qui créent des tables doivent auparavent les effacer avec une clause `DROP TABLE IF EXISTS`. Par exemple, on écrira pour créer la table `Aureus_Creancier` :

```
DROP TABLE IF EXISTS `Aureus_Creancier`;

CREATE TABLE Aureus_Creancier (
  id INT(10) NOT NULL DEFAULT '0',
  Libelle VARCHAR(20) NOT NULL DEFAULT '',
  id_telefact INT(10) DEFAULT NULL,
  siren VARCHAR(15) DEFAULT NULL,
  PRIMARY KEY  (id)
);
```

Pour chaque évolution du schéma ou des données, on placera les scripts dans le répertoire de la version (de la base de données ou du logiciel si la base est liée à un logiciel spécifique). On placera dans ce répertoire un script `all.sql` pour les modifications de schéma ou des données communes à toutes les plates-formes et des scripts `itg.sql`, `prp.sql` et `prod.sql` pour les données spécifiques aux plates-formes.

A noter qu'il n'est pas nécessaire de préciser la base de donnée utilisée dans les scripts par une clause USE car la base de données spécifiée dans le fichier de configuration est utilisée pour exécuter les scripts.

Les versions doivent être de la forme `X.Y.Z` (ou un sous ensemble `X.Y` ou `X`), où X, Y et Z sont des entiers, sans quoi le script de migration ne peut les gérer.

Fichier de configuration

Voici un exemple de fichier de configuration :

```
# encoding: UTF-8

# platform list
PLATFORMS = ['itg', 'prp', 'prod']
# default platform
DEFAULT_PLATFORM = PLATFORMS[0]
# platform where init is forbidden
CRITICAL_PLATFORMS = PLATFORMS[1:]
# charset of the database
CHARSET = 'utf8'
# directory where live SQL scripts (None means directory of the script)
SQL_DIR = None

# Database configuration for environments
CONFIGURATION = {
    'itg': {
        'hostname': 'localhost',
        'database': 'test',
        'username': 'test',
        'password': 'test',
    },
    'prp': {
        'hostname': 'localhost',
        'database': 'test',
        'username': 'test',
        'password': 'test',
    },
    'prod': {
        'hostname': 'localhost',
        'database': 'test',
        'username': 'test',
        'password': 'test',
     },
}
```

C'est un fichier Python qui comporte les propriétés de configuration suivantes :

- `PLATFORMS` : la liste des plates-formes à gérer. Typiquement 'itg', 'prp' et 'prod'. Les scripts de migration seront nommés en conséquence.
- `DEFAULT_PLATFORM` : la plate-forme par défaut (celle qui est utilisée lorsqu'aucune plate-forme n'est passée en ligne de commande). L'expression 'PLATFORMS[0]' indique que l'on prend la première plate-forme de la liste.
- `CRITICAL_PLATFORMS` : la liste des plates-formes sur lesquelles l'option '-i' qui réinitialise la base de données (et efface toutes ses tables) est interdite. L'expression 'PLATFORMS[1:]' indique que l'on prend toutes les plateformes sauf la première.
- `CHARSET` : le nom du jeu de caractères utilisé par la base de données ('utf8' ou 'latin1').

- `SQL_DIR` : le répertoire des scripts SQL de migration. Si cette valeur vaut None, alors les scripts sont cherchés dans le répertoire du script de migration.
- `CONFIGURATION` : un dictionnaire par plate-forme indiquant pour chacune : l'hôte de la base, le nom de la base de données, le nom de l'utilisateur et son mot de passe.

Les lignes de code à la fin du fichier ne servent qu'à afficher la configuration de manière lisible pour le commun des mortels.

Script de migration

Le répertoire sql contient le scripts de migration de base de données, `db_migration.py`, le fichier de configuration de la base de données, `db_configuration.py` ainsi que des scripts de migration d'exemple. C'est typiquement le répertoire que l'on trouve dans un projet pour gérer la migration de la base de données.

Pour obtenir de l'aide sur le script, taper la ligne de commande suivante :

```
$ ./db_migration.py -h
python db_migration.py [-h] [-d] [-i] [-a] [-l] [-u] [-s sql_dir] [-c config]
                       [-p fichier] [-m from] platform [version]
-h          Pour afficher cette page d'aide.
-d          Affiche les scripts a installer mais ne les execute pas.
-i          Initialisation de la base ATTENTION ! Efface toutes les donnees.
-a          Pour installer les scripts de toutes les versions du repertoire.
-l          Pour installer sur la base de donnees locale en mode test.
-u          Pour ne rien afficher sur la console (si tout se passe bien).
-s sql_dir  Le répertoire où se trouvent les fichiers SQL (répertoire du script
            par défaut).
-c config   Indique le fichier de configuration à utiliser (db_configuration.py
            dans le répertoire du script par défaut).
-m from     Ecrit le script de migration de la version 'from' vers 'version'
            sur la console. La valeur 'init' indique que tous les scripts de
            migration doivent être inclus.
platform    La plate-forme sur laquelle on doit installer (les valeurs
            possibles sont 'itg', 'prp' et 'prod'). La valeur par defaut est 'itg'.
version     La version a installer (la version de l'archive par defaut).
```

- L'option `-d` (pour dry run) permet de lister les scripts qui seront passés lors de la migration sans les exécuter réellement.
- L'option `-i` intialise la base de données, c'est à dire qu'elle passe d'abord les scripts du répetoire `init`. Les tables de la base de donnée seront effacées. ATTENTION ! Cette commande peut être dangereuse sur les plateformes de PRP ou PROD par exemple. Cependant, cette option est refusée sur les plateformes listées dans la propriété `CRITICAL_PLATFORMS` de la configuration.
- L'option `-a` passe les scripts de migration pour toutes les versions. On ne doit donc pas passer la version en paramètre avec cette option.
- L'option `-l` passe les scripts en local, c'est à dire sur la base MySQL sur `localhost` avec l'utilisateur `test` et le mot de passe `test`. Cet utlisateur et ce mot de passe sont disponibles dans la configuration par défaut de MySQL. A noter que l'utilisateur `test` doit avoir les droits sur la base de donnée de la configuration.

- L'option `-u` passe les scripts silencieusement (sauf en cas d'erreur).
- L'option `-s sql_dir` indique où se trouvent les scripts de migration SQL. Par défaut, les scripts sont cherchés dans le répertoire du script de migration.
- L'option `-m from` permet de générer la requête SQL de migration de la version `from` vers la version `version` passée en ligne de commande. A noter que cette option est incompatble avec les options `-d`, `-l` et `-p`. Cette option est pratique pour mettre à jour la base de donnée sur des plateformes où le script de migration ne peut pas tourner. Cependant, les tables méta ne sont pas mises à jour et l'on doit connaître la version depuis laquelle on migre la base de données.

Si un fichier `VERSION` se trouve dans le répertoire du script, alors la version vers laquelle on migrera la base est extraite de ce fichier. Dans ce cas une extension `-SNAPSHOT` peut se trouver à la fin de la version et elle est ignorée lors des comparaisons.

Exemples

Pour migrer la base d'ITG vers la version 1.2.3 :

```
./db_migration.py itg 1.2.3
```

Pour afficher les scripts à passer pour migrer la base d'ITG vers la version 1.2.3 sans les exécuter (dry run) :

```
./db_migration.py -d itg 1.2.3
```

Pour intialiser la base de donnée et la migrer vers la version 1.2.3 :

```
./db_migration.py -i itg 1.2.3
```

Pour migrer la base d'ITG vers la dernière version :

```
./db_migration.py -a itg
```

Pour installer la base d'ITG en version 1.2.3 en local :

```
./db_migration.py -l itg 1.2.3
```

Pour générer le script de migration SQL de la base d'ITG de la version 1.0.0 vers la version 1.2.3 :

```
./db_migration.py -m 1.0.0 itg 1.2.3 > migration-1.0.0-1.2.3.sql
```

Installation du script

Pour installer le script dans votre projet, créer un répertoire dédié à la base de donnée (par exemple `sql`) et y recopier le contenu du répertoire sql (en particulier les scripts `db_migration.py` et `db_configuration.py`). On personalisera le fichier de configuration `db_configuration.py`.

Placer ensuite les scripts d'initialisation dans un répertoire `init` et ceux de migration dans des répertoires `x.y.z`. Le script de migration cherche les répertoires des scripts de migration dans son propre répertoire.

On pourra alors livrer le script de migration (avec sa configuration et les scripts SQL qui l'accompagnent) lors des releases. On peut même automatiser la migration des bases à l'aide d'un script de post installation appelant la procédure de migration que l'on aura packagée dans l'archive.

Pré-requis pour faire tourner le script

Pour faire tourner le script, on doit disposer des pré-requis suivants :

- Une version récente de python (le script a été testé avec des versions 2.5.2, 2.6.6 et 2.7.2).
- La commande mysql doit être installée. Elle est utilisée pour exécuter les scripts de migration.

Si la plateforme ne dispose pas de ces pré-requis, il est toujours possible de gérer les migrations en générant des scripts de migration avec l'option `-m` (voir l'option dans la section `Script de migration` ci-dessus).

Tables des méta données

Le script de migration gère deux tables de méta données dans la base :

Table `_scripts`

Elle contient les informations relatives au passage des scripts de migration. Exemple d'une telle table :

```
+--------------+---------------------+---------+------------+---------------+
| filename     | install_date        | success | install_id | error_message |
+--------------+---------------------+---------+------------+---------------+
| init/all.sql | 2012-04-18 14:17:20 |       1 |          1 | NULL          |
| init/itg.sql | 2012-04-18 14:17:20 |       1 |          1 | NULL          |
| 0.1/all.sql  | 2012-04-18 14:17:20 |       1 |          1 | NULL          |
+--------------+---------------------+---------+------------+---------------+
```

Cette table liste les scripts passés (avec le chemin relatif au répertoire du script) avec leur date d'installation, le succès et un éventuel message d'erreur. De plus elle indique la référence de l'installation correspondante dans la table `_install`.

Table `_install`

Elle liste les migrations de la base :

```
+----+---------+------------------+------------------+---------+
| id | version | start_date       | end_date         | success |
+----+---------+------------------+------------------+---------+
|  1 | 0.1     | 2012-04-18 14... | 2012-04-18 14... |       1 |
|  2 | 1.0     | 2012-04-18 14... | 2012-04-18 14... |       1 |
+----+---------+------------------+------------------+---------+
```

La table liste les migrations avec leur version (sous la forme 'major.minor. debug'), les dates de début et de fin de migration ainsi que le succès de la migration.

Comment créer les tables des méta données à la main

Les tables de méta données sont générées automatiquement à l'init (option -i). Néanmoins, si vous devez les créer à la main, voici les instructions :

```
CREATE TABLE IF NOT EXISTS _install (
  id integer NOT NULL AUTO_INCREMENT,
  version varchar(20) NOT NULL COMMENT 'Numero de version de la base',
  start_date datetime NOT NULL COMMENT 'Date de debut d''installation',
  end_date datetime COMMENT 'Date de fin d''installation',
  success boolean NOT NULL COMMENT 'Indicateur de succes de l''installation',
  PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Versionning de la base';

CREATE TABLE IF NOT EXISTS _scripts (
  filename varchar(255) NOT NULL COMMENT 'Nom du fichier SQL installe',
  install_date datetime NOT NULL COMMENT 'Date d''installation du script',
  success boolean NOT NULL COMMENT 'Indicateur de succes de l''installation',
  install_id integer NOT NULL COMMENT 'ID de l''installation en cours',
  error_message text COMMENT 'Error message, null if script was successful',
  CONSTRAINT fk_install_id
    FOREIGN KEY (install_id)
    REFERENCES _install(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Historique de passage des scripts';
```

Enjoy!